TABLE OF CONTENTS

USING LEVEL III BASIC

INTRODUCTION TO LEVEL III BASIC

If you think there are significant differences between Radio Shack's Level I and Level II Basics, be prepared for another quantum jump in personal programming!

G2's LEVEL III BASIC is not just another Basic. It is an actual enhancement to the already powerful Level II Basic. It adds the features of Radio Shack's Disk Basic to Level II and you don't have to buy an expansion box and a disk drive. And it has many more great features including advanced graphics that turn your TRS-80 screen into a virtual electronic drawing board.

G2's LEVEL III BASIC is for TRS-80 computers with a minimum of 16K Ram memory and Level II Basic. It comes on cassette tape and resides in less than 5.25K Ram memory. Even with the minimum system, this leaves you with more than 10K Ram for programming space!

You'll soon discover that LEVEL III BASIC also corrects some of the problems with Level II Basic. It eliminates keyboard bounce and provides more reliable cassette tape loading. Two new commands, LOAD and SAVE, replace the Level II CLOAD and CSAVE commands. They permit cassette tape interfacing without the usual "volume sensitivity" of the recorder.

G2's LEVEL III BASIC has a NAME command that will automatically renumber program line numbers, and do it the way you specify. For example, you can renumber a program beginning at line number 150 and increment it by 25 (150, 175, 200, etc.). Line numbers following GOTOs, GOSUBs, etc., are automatically edited to reflect the new sequence. With this feature, you'll find it much easier to edit programs. Your programs will be better organized and you won't have to worry about running out of space between program lines.

Another major feature of LEVEL III BASIC is a set of 26 **user-changeable** "Shift-Key Entries" Just press the [SHIFT] key and the [R] key simultaneously and LEVEL III will execute a RUN command. You don't have to type R-U-N. You don't even have to use [ENTER] ! Call it the lazy way to computer programming if you like, but you'll soon find yourself using this feature.

Other features of LEVEL III BASIC include Error Messages that are "spelled out" instead of abbreviated; a new input instruction called LINE INPUT that allows punctuation in an input response; an INSTR instruction to search a string for a substring; user defined functions; automatic Hex and Octal conversion; and much more.

As you can see, G2's LEVEL III BASIC is bound to broaden your programming horizons. It brings ease, speed, power and flexibility to your TRS-80 Level II Basic computer that you might have thought not possible. So dig in and start programming!

LEVEL III AUTHOR

In 1975, Microsoft wrote Altair Basic, the first Basic interpreter for a microcomputer. Since then, Microsoft-written Basics have become the virtual standard of the personal computing industry.

Microsoft has a reputation for quality, advanced programming techniques and efficiency that is surpassed by none. LEVEL III BASIC was written by Microsoft President, Bill Gates. GRT acknowledges and thanks Bill Gates for this significant contribution to the G2 Software Library.

ABOUT THIS BOOKLET

By necessity, the authors of this instruction booklet have assumed the readers have some experience programming in Basic and in particula Radio Shack's Level II Basic.

Still, we do not expect you to be a programming expert. Every descrip tion has been carefully worded to avoid undue technicality. Example are provided throughout.

This instruction booklet began as a comprehensive reference manua written by Andrea Lewis. It was then edited and rewritten into its currer form by David Bunnell. GRT acknowledges and thanks Andrea Lewi and David Bunnell for their contribution to G2 Software Documentatior

PROGRAM SPECIFICATIONS

Basic Interpreter Required—In order to use G2's LEVEL III BASIC you must have a Radio Shack TRS-80 Computer equipped with Radio Shack's Level II Basic Interpreter.

Minimum Memory Requirement—Your TRS-80 Computer must have a minimum of 16K RAM memory to run this software.

Tape Format—All G2 cassette software is recorded on tape selected for its response and suitability for recording computer software. There are four recordings on Side One of G2's LEVEL III BASIC tape. These recordings and the order in which they are recorded include: **1)** LEVEL III BASIC (Cassette File), **2)** LEVEL III BASIC (Cassette File), **3)** LEVEL III BASIC (Disk File), and **4)** LEVEL III BASIC (Disk File).

3. Connect the gray cable from the Video Monitor to the VIDEO jack on the back of the Keyboard. Do this carefully, making sure that the pins line up correctly.
4. Using the same care as above, connect the gray cable from the Power Supply to the back of the Keyboard.
5. Connect the short cassette cable to the TAPE jack on the back of the Keyboard.
6. Connect the **black** plug on the other end of the cable to the EAR jack on your recorder.
7. Connect the large gray plug to the AUX jack on your tape recorder and the smaller gray plug to the REM jack. These two connections are not necessary for loading programs from tape.
8. After double-checking all connections, turn on the POWER switch on the back of the Keyboard and the POWER switch on the front of the Video Monitor.
9. For best results, allow your system to warm up for a few minutes before attempting to load LEVEL III BASIC or any other programs.

LOADING G2's LEVEL III BASIC (CASSETTE FILE)

Follow these instructions to load:

1. Insert the LEVEL III BASIC cassette into your tape deck.
2. Press the [REWIND] key on your tape deck, and allow the tape to rewind fully.
3. Type the word SYSTEM and press [ENTER]. Your computer should respond with the prompt: *?
4. Following the prompt, type the word LEV3 and press [ENTER].
5. Press the [PLAY] key on your tape deck.
6. If the software is loading correctly, two asterisks (*) will appear in the upper-right-hand corner of your video screen. The asterisk (*) on the right should blink on and off. LEVEL III takes approximately two minutes to load.
7. When the program is loaded, the prompt (*?) will once again appear. Press the slash [/] key followed by [ENTER]. If LEVEL III BASIC has been loaded correctly, the screen will display:

```
LEVEL III CASSETTE BASIC
A G2 PRODUCT FROM GRT
COPYRIGHT MICROSOFT 1979
READY
>
```

8. As soon as the program has been loaded, press the [REWIND] key on your tape deck. Let the tape rewind, then stop the tape deck and remove the cassette. This will protect the cassette from any harm, no matter what you do from this point on.
9. If you have trouble getting a good load, see "IN CASE OF LOADING TROUBLES" on page 10.

LOADING LEVEL III DISK BASIC (DISK FILE)

The third and fourth recordings on Side One of the tape are identical copies of the Disk File version of LEVEL III BASIC. This version of LEVEL III is for people who have TRS-80 Disk Systems and want to save LEVEL III on diskette. We do not mean to imply by this that LEVEL III will work with Disk BASIC. As an enhancement to Level II Basic, LEVEL III works only with Level II Basic.

To save LEVEL III on diskette, you need to have the TAPEDISK utility, which is included with most Radio Shack Disk Operating Systems. If you do not have this utility, check with your Radio Shack dealer.

Read the following instructions to load LEVEL III BASIC (Disk File) and save it on diskette:

1. Insert the LEVEL III BASIC cassette into your tape recorder. You can rewind the tape to its beginning or you can cue it to the third or fourth recording.
2. Turn on your TRS-80 System. The screen should display: DOS READY.
3. Type TAPEDISK [ENTER].
4. The screen should display a question mark (?) as a prompt. Following this prompt, type: C DLEV3 [ENTER].
5. Press the [PLAY] key on your tape deck.
6. If the software is loading correctly, a **single** blinking asterisk (*) will appear in the upper-right-hand corner of the screen.
7. Once the program has been successfully loaded, the ? prompt will again appear.
8. Type F DLEV3/CMD: 0 5500 6A00 5500 [ENTER].
9. The first zero (0) in the above code is the Drive number. You can save LEVEL III in any existing Drive, but you must enter the corresponding number.
10. You should hear the "grinding" sound of the Disk Drive as LEVEL III is saved. Once this operation is completed, the ? prompt will again appear. LEVEL III BASIC has been successfully saved on diskette.

11. To use LEVEL III once it is saved on diskette, do the following:
    - —Turn the computer on. When it says, DOS READY, type BASIC [ENTER]. The screen will display two questions: FILES?, followed by SIZE OF MEMORY? Answer both with the [ENTER] key.
    - —Once Disk Basic is up and running type CMD"S" [ENTER]. This will get you back to DOS so you can load the LEVEL III file.
    - —Type DLEV3 [ENTER]. LEVEL III should come up on the screen and you are ready to program.

IN CASE OF LOADING TROUBLES

Your TRS-80 may be very "volume sensitive" when it comes to loading cassette tapes. While LEVEL III BASIC eliminates this problem with its new LOAD and SAVE commands, you still have to load LEVEL III using the LEVEL II SYSTEM command. This is a common frustration, but it can usually be alleviated by trying one of the following:

1. Recheck all power lines and interconnecting cords.
2. Remove cable from the earphone jack of your tape deck, and play the tape. You should hear both the leader tones and the digital program signals clearly. If you do not hear these sounds, your tape deck may be faulty. Try it with a different tape deck.
3. The most common loading problems are due to improper settings of the **tone** and **volume controls** on your tape deck. When loading cassettes, adjust your volume control up one half level at a time. The tone control is less critical, but adjust it also.

   Once you have successfully loaded a tape, jot down the correct settings on the cassette label.

   Since no two tape decks have exactly the same characteristics, it is impossible to give you specific machine settings. But a little trial-and-error will establish the best settings.
4. If you still cannot load a tape, you may need to clean the head on your cassette recorder. Use a high quality head cleaner, one with a proper cleaning pad. Watch out for cleaner-tapes, they are often abrasive and have been known to scratch a head or two!

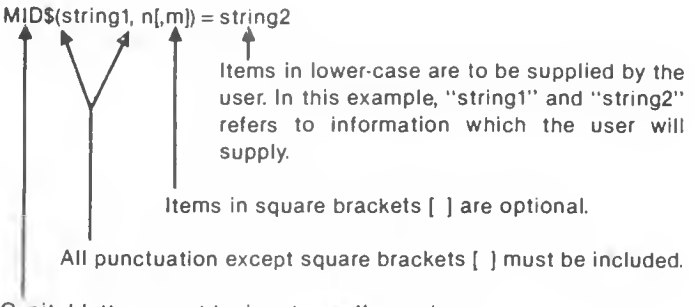
   There are many sources for head cleaning kits including Radio Shack.
5. Demagnetize the head on your tape recorder. Through continuous use, the head builds up a thin layer of magnetism which affects its ability to accurately read tapes. You can buy

a simple device called a "Head Demagnetizer" for about $5 and up that does this task. Ask your Radio Shack dealer or electronics dealer for a demonstration.

6. Try running the tape through Rewind and Fast Forward a few times. This will remove dust or other material on the tape that is preventing a load.
7. Let your computer warmup for a few minutes before attempting to load programs.
8. Try a different cassette recorder.
9. Problems can result from fluctuations in your power lines, though this is rare. As a precaution, try restricting the TRS-80 to its own circuit. If possible, even run your cassette recorder and video monitor on a separate line.
10. Ask your Radio Shack dealer about their "cassette modification" fix for Level II Basic computers. This is a hardware fix that might be covered by your TRS-80 warranty. It should make the volume control on your recorder less sensitive.
11. If you are still not having much success, discuss the problem with your local computer dealer.

FORMAT NOTATION

To aid you in reading and understanding this instruction booklet, a **format notation** has been devised for introducing statements, commands, and functions. An example of this notation is as follows:



The following rules apply to notation:

1. Items in capital letters must be input **exactly** as shown.
2. Items in lower case letters are to be supplied by the user.
4. Items in square brackets [ ] are optional.
5. All punctuation (i.e., commas, parentheses, semicolons, hyphens, equal signs) except square brackets [ ] must be included where shown.
6. All blank spaces are optional, unless otherwise noted.

USING LEVEL III BASIC

SECTION ONE: SPECIAL FEATURES

INCLUDING:

- SHIFT-KEY ENTRIES
- WRITING YOUR OWN SHIFT-KEY ENTRIES
- RENUMBERING PROGRAM LINES
- SPELLED-OUT ERROR MESSAGES
- LEVEL III'S DIGITAL CALENDAR-CLOCK*
- CONVERTING HEX AND OCTAL
- LOCKOUT RECOVERY

The "Special Features" of LEVEL III BASIC are presented here first because some of them can be of immediate use. This is particularly true of the time-saving technique called Shift-Key Entries which allow you to enter instructions or string expressions by simultaneously pressing [SHIFT] and a letter key (A-Z). You'll also want to know about Renumbering Program Lines and about Spelled-Out Error Messages before you go on.

*Requires that your TRS-80 include an Expansion Box

SHIFT-KEY ENTRIES

With Shift-Key Entries you can instantly enter Basic instructions and frequently used string expressions. These might include common responses to INPUT statements or just about any frequently used phrase or formula. All you have to do is hold down [SHIFT] and type in a single letter from A to Z.

For example, [SHIFT] [P] instantly enters the instruction, PRINT.

LEVEL III BASIC contains a complete list of 26 Shift-Key Entries. These include mostly Basic statements and commands.

For added convenience, the commands CONT, EDIT, and RUN are combined with [ENTER]. In other words, if your enter [SHIFT] [C], the computer will respond with CONT (ENTER). You will **not** have to press the [ENTER] key. Other Shift-Key Entries include punctuations such as left parentheses and leading quotation marks.

LSET LIST. LEVEL III BASIC maintains an updated list of Shift-Key Entries that you can display on your screen by typing LSET LIST and pressing [ENTER]. If you change any of the Shift-Key Entries (see following section), this list will be automatically updated.

For your added convenience, they are also listed here as follows:

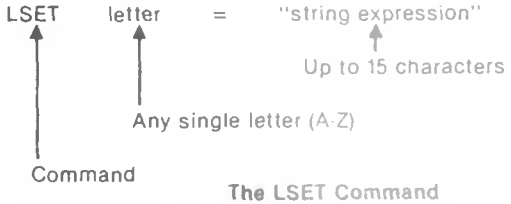
LEVEL III BASIC
Shift-Key Entries

SHIFT Key	Sequence
A	AUTO
B	GET@(
C	ELSE
D	EDIT↓
E	EDIT
F	GOTO
G	GOSUB
H	INKEY$
I	INPUT
J	LINE INPUT
K	LINE(
L	LIST
M	LSET
N	NEXT
O	PRINT USING
P	PUT@(
Q	RETURN
R	RUN↓
S	SAVE"
T	THEN
U	TIME$
V	LOAD"
W	LEFT$(
X	MID$(
Y	RIGHT$(
Z	STRING$(

NOTE. The downward arrow (↓) is the symbol in this chart for [ENTER].

WRITING YOUR OWN SHIFT-KEY ENTRIES

You can use the LSET Command to write your own Shift-Key Entries. The format of this command is as follows:



The LSET Command

The "string expression" in an LSET command can be any string of up to 15 characters. This makes it possible to use Shift-Key Entries for any occassion where a repeated phrase or formula is entered into the computer. In addition to storing frequently used statements and commands, Shift-Key Entries are often used to store "input responses."

Example. When writing programs that use LEVEL III graphics, you frequently find yourself typing in the same first five characters of a LINE statement, namely LINE(. You could make LINE(a Shift-Key Entry with the following:

LSET L = "LINE(" ENTER

From this point on, LINE(will replace LIST in the "Shift-L sequence." Press SHIFT L and your TRS-80 screen will display:

LINE(

To change the Shift-L sequence back to LIST you simply reverse the process by entering:

LSET L = "LIST" ENTER

Note. Loading LEVEL III will always bring up the orginal Shift-Key Entries.

Example. Some programs may require you to repeatedly input phrases such as "TOTALS," "AVERAGE TO DATE," "SALES TAX," etc. You can save typing time by turning these phrases into Shift-Key Entries, such as:

LSET T = "TOTALS" [ENTER]

LSET A = "AVERAGE TO DATE" [ENTER]

LSET S = "SALES TAX" [ENTER]

Press [SHIFT] [T] and the word TOTALS will appear on the screen. [SHIFT] [A] returns AVERAGE TO DATE, and [SHIFT] [S] returns SALES TAX.

Shift-Key Entries with ENTER. The downward arrow (↓) is used in the LSET list as a symbol for ENTER. However, it is not recognized as such by Basic. Pressing [↓] will cause the cursor to move down to the next line.

To add ENTER to one of your own Shift-Key Entries, you use the CHR$ string with the ASCII value for ENTER (13). Technically speaking, you concatenate the ENTER key on the rest of the string. The format is as follows:

LSET R = "RUN" + CHR$ (13) [ENTER]

LSET RESET. You can "turn off" all the Shift-Key Entries by typing LSET RESET, and pressing [ENTER]. The purpose of this command is to free shift-letters for possible use with printers that have lower-case letters.

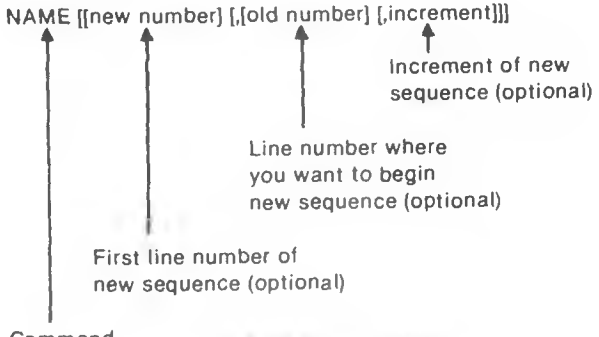
LSET SET. This command will turn Shift-Key Entries back on again.

Note. Available Ram memory is **not** affected in any way by using **LSET SET** or **LSET RESET.**

RENUMBERING PROGRAM LINES

Renumbering program lines is a very useful feature that eases the path to programming excellence. You can renumber to "make room" in a program sequence where additional lines need to be inserted **or** you can use it to better organize program lines.

The NAME Command. In LEVEL III BASIC, the NAME Command is used to renumber program lines. It is a really flexible command, as you can see by its format:



The NAME Command

The NAME command automatically updates all the line number references following GOTO, GOSUB, THEN, ELSE, RESUME, INPUT#LEN, ON...GOTO, and ON...GOSUB. The **first time** you execute a NAME command these particular line numbers are allotted a total of five spaces each. This creates a gap in some statements such as, 10 GOTO 50, 80, 100. On subsequent executions of NAME, this gap remains but is not again increased in size.

From zero to three numbers can follow the NAME command. They are optional and include the following:

—**New number.** This refers to the first line number in a new sequence. If you don't include it, the first number will be 10.

—**Old number.** This is the current line number where you want the new sequence to begin. If, for example, you wanted a program

to be renumbered beginning at line number 500, you would put 500 at this location. If you don't include an "old number," the complete program will be renumbered from its beginning line number.

— **increment.** This is the increment you want in the new sequence. If you don't use it, the program will be renumbered in a sequence of 10 (10, 20, 30, 40, 50, etc.).

Examples. The three optional numbers following the NAME command make it very versatile. You can use it to renumber program lines in any of the following ways:

Command	Explanation
NAME	Renumbers the entire program. The first line number will be 10 and each following line number will be incremented by 10 (10, 20, 30, etc.).
NAME 100	Renumbers the entire program. The first line number will be 100 and each following line number will be incremented by 10 (100, 110, 120 etc.).
NAME 100, 50	Renumbers program **beginning at line 50**. Line 50 will become line 100. Each following number will be incremented by 10 (100, 110, 120, etc.).
NAME 100, , 50	Renumbers entire program. The first line number will become 100 and following line numbers will be incremented by 50. (100, 150, 200, etc.).
NAME , 100	Renumber program beginning at existing line 100. Line numbers following 100 will be incremented by 10 (100, 110, 120, etc.).

NAME , , 100	Renumbers entire program. The first line number will become line 10 and the following numbers will be incremented by 100 (10, 110, 210, etc.)
NAME , 500, 20	Renumbers program beginning with existing line number 500. Increments by 20 (500, 520, 540, etc.).
NAME 100, 50, 20	Renumbers the program—beginning at existing line number 50. Line 50 will become line 100. Increments by 20 (100, 120, 140, etc.)

Note. NAME cannot be used to **change the order of program lines.** Also, it will **not** create a line number greater than 65529. An ILLEGAL FUNCTION CALL error will result if either of these is attempted.

If you omit "THEN" from an IF...THEN statement the following line number will not change when you use NAME. In the statement, 30 IF A$ = "YES" 60, the number 60 will not be changed. However, in the statement, 30 IF A$ = "YES" THEN 60, the number will be changed accordingly.

SPELLED-OUT ERROR MESSAGES

Level III saves you the trouble of looking up error code abbreviations by printing out the complete error message. Instead of displaying the Level II code:

?TM ERROR IN 10

for a Type Mismatch error, Level III displays:

TYPE MISMATCH IN 10

All Level III error messages are listed on your **Level III Basic Reference Card**. For convenience, they are also listed here as follows:

LEVEL III ERROR MESSAGES

Error Code	Level II Abbreviation	Level III Error Message
1	NF	NEXT without FOR
2	SN	Syntax error
3	RG	Return without GOSUB
4	OD	Out of data
5	FC	Illegal function call
6	OV	Overflow
7	OM	Out of Memory
8	UL	Undefined Line
9	BS	Subscript out of range
10	DD	Redimensioned array
11	/0	Division by zero
12	ID	Illegal direct
13	TM	Type mismatch
14	OS	Out of string space

Error Code	Level II Abbreviation	Level III Error Message
15	LS	String too long
16	ST	String formula too complex
17	CN	Can't Continue
18	NR	No RESUME
19	RW	RESUME without error
20	UE	Unprintable error
21	MO	Missing operand
22	FD	Bad file data
23	L3	Disk Basic only

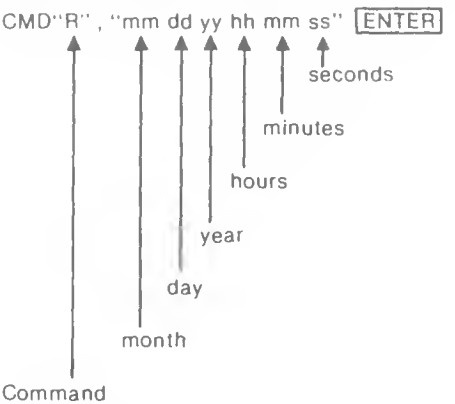
More Information. For further explanation of these error messages, refer to Appendix B, pages B/2-B/3 of your Radio Shack **LEVEL II BASIC Reference Manual.*** If you don't have a copy of this publication, check with your Radio Shack dealer.

LEVEL III'S DIGITAL CALENDAR-CLOCK *

LEVEL III stores a string, TIME$, that keeps track of the date and time, providing your TRS-80 Computer system includes a Radio Shack Expansion Box. This accessory is necessary because TIME$ requires a Real Time Clock, a feature of the Expansion Box.

TIME$ can be utilized like any other string. It can be used in any program where timing is useful, or where you might want to display the date.

The CMD "R" Command. When you load LEVEL III BASIC, the TIME$ string contains all zeroes. You can set it to the correct time and date or to any time and date you want by using the CMD"R" command. The format of this command is as follows:



Notice that you are not required to enter dashes or commas between each number. The format for TIME$ has been done for you and it is fixed.

*Requires Radio Shack's TRS-80 Expansion Box

Each entry requires a two digit number. If the date is May 7, enter 07 for day.

The hh (hours) is based on a 24-hour clock which includes hours 01 to 24. The morning hours (a.m.) are 01 to 12, while the afternoon hours are 13 to 24.

Example. To set the date, July 25, 1979, and the time, 2:15 p.m., enter:

CMD"R" , "07 01 79 14 15 00" ENTER

leave one space

Regardless of whether you set the Digital Clock-Calendar or not, it begins keeping track of the time and date the instant LEVEL III BASIC is loaded. All settings start at 00.

Hints on Using TIME$. You can use TIME$ with RIGHT$, LEFT$, or MID$ to display one or any combination of the elements in the Digital Clock-Calendar. For example, the instruction, PRINT LEFT$(TIME$, 8) will return just the date. Other examples include:

Instruction	Explanation
PRINT RIGHT$(TIME$, 2)	Returns seconds
PRINT MID$(TIME$, 10, 2)	Returns hours
PRINT MID$(TIME$, 13, 2)	Returns minutes
PRINT RIGHT$(TIME$, 8)	Returns time

The computer will recognize the above "Returns" as string elements, not numbers. Therefore, you cannot use these commands to make comparisons. In order for the computer to recognize the returns as numbers, you need to use the VAL function. For example:

```
10 X = VAL (MID$(TIME$, 10, 2))
20 IF X<12 THEN 80
```

HEXADECIMAL AND OCTAL CONSTANTS

Level III includes a routine that automatically converts hexadecimal (base 16) and octal (base 8) numbers into decimal numbers.

Thus, hexadecimal and octal numbers can be supplied directly to Basic programs, which always require decimal numbers. You don't need to do the conversion. Simply prefix hexadecimal numbers with [&] [H] and octal numbers with [&].

This will work with all Basic statements and commands **except** DATA statements or in response to INPUT statements.

You can use this feature as a simple conversion calculator by using the PRINT command. For example, to convert the hexadecimal number 4A5F to decimal, enter:

```
PRINT &H4A5F
```

and the computer will answer with: 19039.

Example. If you want to POKE the hex value FF (255 decimal) into location 4A5F (19039 decimal), use the statement:

```
POKE &H4A5F, &HFF
```

A subsequent PEEK at location 19039 will return the value 255.

LOCKOUT RECOVERY

A System Lockout is when you lose control of your computer. It just sit there and "smiles" at you. No matter what key you press, nothing hap pens.

In Level II Basic, you can get out of this problem by pressing the RESE button in the back, upper-left-hand corner of the keyboard cas However, if you have an Expansion Box this will result in a complete los of your program from memory.

LEVEL III BASIC eliminates this problem by allowing you to use th [BREAK] key just as you would use RESET. Using the [BREAK] ke even with an Expansion Box, will not result in the loss of a program. Th system will start up again, and your program will be preserved.

USING LEVEL III BASIC

SECTION TWO: GRAPHICS

INCLUDING:

- CHARACTER MODE
- GRAPHIC MODE
- LINE STATEMENT
- EXAMPLES OF GRAPHICS PROGRAMS
- GET@ STATEMENT
- PUT@ STATEMENT
- NOTES ON USING GET@ AND PUT@
- DIMENSIONING GRAPHIC ARRAYS
- MORE EXAMPLES OF GRAPHICS PROGRAMS
- EXAMPLES OF ADVANCED GRAPHICS

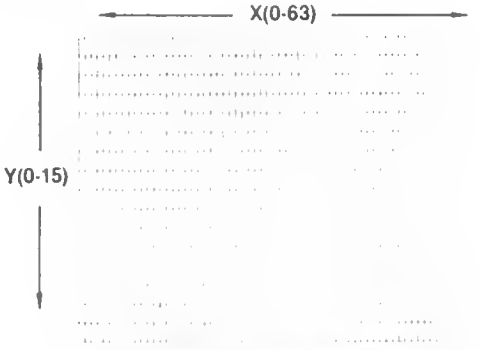
If you've had any experience at all working with Level II graphics, you will immediately recognize the expanded possibilities that LEVEL III graphics give you.

LEVEL III graphics will generate a line or rectangle between any two points on the screen. This frees you from the tedious task of defining all the points yourself, as you must do in Level II Basic.

LEVEL III graphics also let you store a graphic array for later use on the same or a different portion of the screen. With this added capability, you'll find yourself writing more programs with graphs, pictures and even animation.

CHARACTER MODE

There are two modes of graphics presentation called Character Mode and Graphics Mode. In the first of these, your TRS-80 screen is divided into a grid that measures 64 "character positions" across by 16 character positions down.

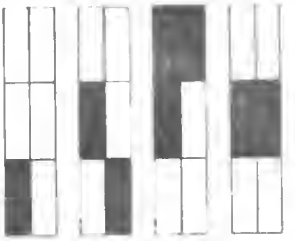


SCREEN (Character Mode)

Each character position in Character Mode is further broken down into a two-by-three grid of "graphic blocks" that look something like this:

When you are in Character Mode, you can put any alpha numeric character on the screen at any specified character position. Also, you can use any one of a set of special "graphic symbols."

There are 64 graphic symbols made possible by filling different combinations of graphic blocks in a character position, such as



Using CHR$. To specify a particular graphic symbol, the string expression CHR$ is used. This expression will return any alpha-numeric character or graphic symbol when the appropriate ASCII value is supplied with it, such as CHR$(147).

Each alpha-numeric character has been assigned an ASCII value of from 32 to 127, while each graphic symbol has a value of 128 to 191.

The ASCII values and their corresponding alpha-numeric characters are listed on page C/2 of your **LEVEL II BASIC Reference Manual**. The following program will display the graphic symbols with their corresponding number:

```
10 FOR X = 129 to 191
20 PRINT X; :PRINT CHR$(X)
30 NEXT
```

Instead of running the above program each time you want the value for a specific symbol, you can determine the value with the following:

1. Assume that each graphic block has the following value:

2. The code for any graphic symbol is 128 **plus** the total value of the graphic blocks that are to be "turned on." CHR$(128) returns a blank space. CHR$(129) turns on the upper-left-hand graphic block. CHR$(128 + 11) looks like this:

Character Mode Coordinates. When using the Level II Basic statement, PRINT@, each character position is assigned a unique number from 0 to 1023. These positions are illustrated by the "TRS-80 Video Display Worksheet" on page E/1 of your **LEVEL II BASIC Reference Manual**.

Thus, if you wanted to display the message, "THE GAME IS UP," in the middle of the screen, you could do so with the following:

PRINT@ 468, "THE GAME IS UP" ENTER

However, when using LEVEL III BASIC graphic statements (LINE, GET@, and PUT@) in Character Mode, the "X" coordinates (0-63) and the "Y" coordinates (0-15) are used to define character positions. Position 468 is referred to as "20, 7" where the X coordinate is 20 and the Y coordinate is 7.

You can convert from character position numbers to "X" and "Y" coordinates by using the following formulas:

X = N – (64 ★ Y)

Y = INT(N/64)

In the above example, character position 468 would be converted as follows:

X = 468 – (64 ★ 7)	Y = INT(468/64)
X = 468 – 448	Y = INT(7.31)
X = 20	Y = 7

To convert the other way, from coordinates to character position number, use:

N = (Y ★ 64 + X)

Using the same example:

N = (7 ★ 64 + 20)

N = 448 + 20 = 468

The most important thing to remember is simply that in Character Mode the screen is 64 across by 16 down.

GRAPHICS MODE

The second mode of graphic presentation is called the Graphics Mode. In this mode, the screen is divided into a much finer grid using the graphic-blocks described above. The grid is 128 across by 48 down:

SCREEN (Graphics Mode)

The Level II Basic functions SET, RESET and POINT use the screen in its Graphics Mode. Thus, if you want to turn-on graphic-block 10,29, you can do so by entering:

```
10 SET (10,29)
RUN  ENTER
```

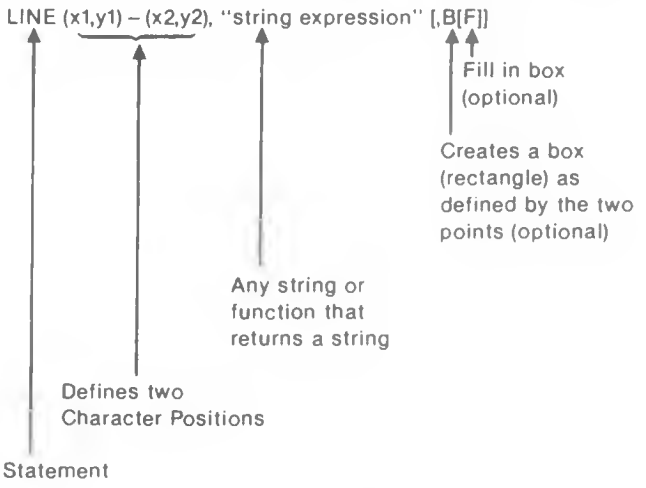
For more information on SET, RESET and POINT see pages 8/1-8/2 of your Radio Shack **LEVEL II BASIC Reference Manual**.

The graphic statements LINE, GET@ and PUT@ can be used in either Character or Graphics Mode. Descriptions of their uses are found in the following pages.

THE LINE STATEMENT

The LINE statement can be used to draw a line between any two points on the screen or to draw a rectangle, assuming the two points are opposite corners of the rectangle. It can be used in both the Character Mode and the Graphics Mode.

The format of the LINE statement in **Character Mode** is as follows:



LINE Statement
(Character Mode)

When the LINE statement in Character Mode is executed, a line or rectangle is drawn between points (x1,y1) and (x2,y2) **using the first character of the string expression**. Thus, if the string expression is "XEO GEE GOLLY," the line or rectangle will be drawn using the X character only.

The string expression may be a string literal as above, **or** it may be a string variable (such as A$), **or** it may be a **function that returns a string** (such as CHR$(142)).

The statement, 10 LINE (32,14)-(45,10), CHR$ (129), would generate a line between character positions (32,14) and (45,10) using the graphic symbol described by CHR$ (129).

True lines can only be generated if they are vertical or horizontal. Executing the following statement:

```
10 LINE (32,14)-(45,10),"X"
```

will result in a line that looks like this:

```
            X
          XXX
        XXX
      XXX
    XXXX
```

If the optional B is included in a LINE statement, a box (rectangle) defined by the two points is drawn. Adding "B" to the above example:

```
10 LINE (32,14)-(45,10),"X",B
                         ↑
                    don't forget
                    the comma
```

will result in a rectangle that looks like this:

```
XXXXXXXXXXXXXXX

X              X

X              X

X              X

XXXXXXXXXXXXXXX
```

If the optional BF is included, a filled-in rectangle is drawn. Adding BF to our example:

```
10 LINE (32,14)-(45,10),"X",BF
```

results in the following:

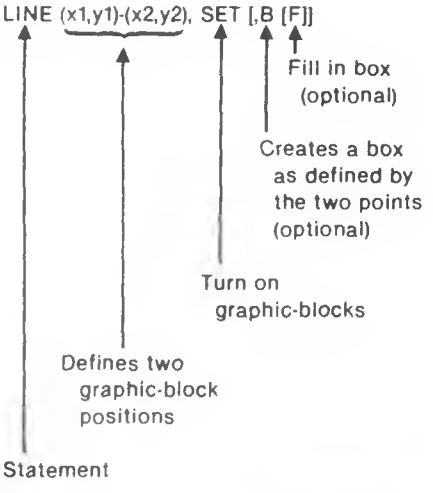
```
XXXXXXXXXXXXXXX

XXXXXXXXXXXXXXX

XXXXXXXXXXXXXXX

XXXXXXXXXXXXXXX

XXXXXXXXXXXXXXX
```

Obviously, you cannot draw a rectangle between two horizontal or two vertical points. The points have to define **diagonally opposite corners** of the rectangle.

To clear the screen of the above graphic, a blank string (" ") can be used such as:

```
20 LINE (32,14)-(45,10)," ",BF
```

The LINE statement can also be used in the **Graphics Mode**. The format of the LINE statement in the Graphics Mode is:



LINE Statement (Graphics Mode)

When the LINE statement in Graphics Mode is executed, a line or rectangle is drawn between points (x1,y1) and (x2,y2) **by turning on the appropriate graphic-blocks.**

As in the Character Mode, the B and the BF are optional. Adding a "B" to a LINE statement results in a box (rectangle) defined by the two points. Adding a "BF" results in a filled-in box.

The word SET in a LINE statement can be replaced with the word RESET. RESET will **turn-off or erase** the line or rectangle described by the statement.

EXAMPLES OF GRAPHICS PROGRAMS

HAVING FUN WITH BOXES

The following program will draw a filled-in box on the screen, leave it for a few seconds, then erase it:

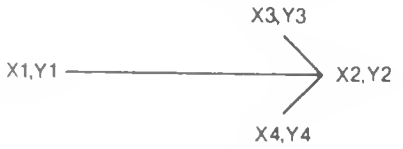
Program	Explanation
10 CLS	clears screen
20 LINE (10,33)-(18,39),SET,BF	turns on box
30 FOR Z = 1 TO 350: NEXT	timing loop
40 LINE (10,33)-(18,39),RESET,BF	turns off box

A few simple modifications to the above program will make the box "animated". It will appear to move across the screen:

Program	Explanation
10 CLS	clears screen
20 X = 10:X1 = 18:Y = 33:Y1 = 39	assigns value
30 LINE (X,Y)-(X1,Y1),SET,BF	turns on box
40 FOR Z = 1 TO 100: NEXT	timing loop
50 LINE (X,Y)-(X1,Y1),RESET,BF	turns off box
60 X = X + 4:X1 = X1 + 4	assigns new values
70 GOTO 30	program loop

HAVING FUN WITH ARROWS

Using essentially the same techniques as above, we can define the points of an arrow and make it appear to move across the screen:



THE ARROW

Program	Explanation
10 CLS	clears screen
20 X1 = 4:Y1 = 26:X2 = 27:Y2 = 26:X3 = 21: Y3 = 23:X4 = 21:Y4 = 29	assigns values
30 LINE(X1,Y1)-(X2,Y2),SET	draws arrow
40 LINE(X3,Y3)-(X2,Y2),SET	
50 LINE(X4,Y4)-(X2,Y2),SET	
60 FOR Z = 1 TO 100:NEXT	timing loop
70 LINE (X1,Y1)-(X2,Y2),RESET	turns off arrow
80 LINE (X3,Y3)-(X2,Y2),RESET	
90 LINE (X4,Y4)-(X2,Y2),RESET	
100 X1 = X1 + 3:X2 = X2 + 3:X3 = X3 + 3: X4 = X4 + 3	assigns new values
110 GOTO 30	program loop

GRAPHIC SYMBOL

This program will alternately fill the screen with the graphic symbols described on page 29.

Program	Explanation
10 CLS	clears screen
20 FOR I = 129 TO 191	assigns values
30 LINE (0,0)-(63,15),CHR$(I),BF	fills screen
40 FOR J = 1 TO 100:NEXT J	timing loop
50 NEXT I	loops back to 20

CRAZY LINES

This program creates patterns of "crazy" lines on the screen:

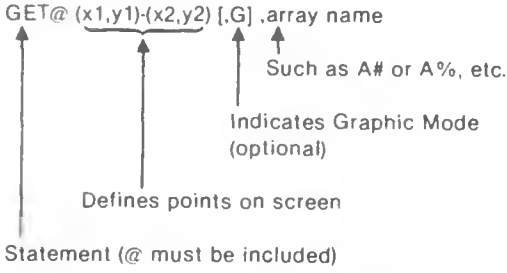
Program	Explanation
10 FOR N = 1 TO 10	assigns values (10 patterns)
20 CLS	clears screen
30 Y1 = 0:X1 = 0	defines point (upper-left corner of screen)
40 FOR I = 1 TO 30	assigns values (30 lines)
50 X2 = RND(127):Y2 = RND(47)	defines second point as any random point on screen
60 LINE (X1,Y1)-(X2,Y2),SET	draws line
70 X1 = X2:Y1 = Y2	redefines first point as second point
80 NEXT I:NEXT N	loops back to 10

Variation: To change this program to create its pattern of lines on a white background, add line 25 LINE (0,0)-(127,47),SET,BF and change SET to RESET in line 60.

GET@ STATEMENT

Often it is useful to save the graphics that are displayed on the screen to reuse later on the same—or different—portion of the screen. You can do this with GET@ and PUT@ statements. The GET@ statement "gets" an area of graphics from the screen and saves it in an array. The PUT@ statement "puts" a graphics array back on the screen.

The format of a GET@ statement is:



GET@ Statement

When the GET@ statement is executed, the graphics that are **currently on the screen** and positioned inside the box defined by (x1,y1)-(x2,y2) are saved into the array. For example:

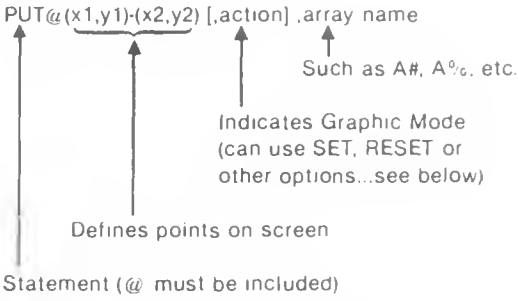
```
10 GET@ (10,4)-(20,9),A%
```

saves the graphics that are in the box defined by (10,4) and (20,9) in the array, A%. (Don't forget to dimension your arrays! For more information, see Chapter 6 of your Radio Shack **LEVEL II BASIC Reference Manual** or page 44 of this booklet.)

The optional [,G] in a GET@ statement determines the mode of the X,Y coordinates. If it is included, the coordinates are assumed to be Graphics Mode coordinates (i.e. X = 0 to 127, Y = 0 to 47). If G is not included, the coordinates are assumed to be in the Character Mode.

PUT @ STATEMENT

Once you have saved the graphics in an array with a GET @ statement, you can put the array back on the screen at any location with a PUT@ statement. The format of PUT@ is:



PUT@ Statement

When the PUT@ statement is executed, the graphics in the named array are put on the screen inside the box defined by (x1,y1) and (x2,y2). For example:

```
10 PUT@ (15,4)-(25,9),A%
```

puts the graphics saved in the array, A%, on the screen inside the box defined by (15,4)-(25,9).

If no [,action] is specified, the coordinates are assumed to be in Character Mode. When an [,action] is specified, the coordinates are assumed to be in Graphics Mode. The "action" of a PUT@ statement may be one of the following:

SET	Puts the array on the screen exactly as it was saved. All the "on" positions are turned on, and all the "off" positions are turned off
RESET	Puts the complement of the array on the screen. All the "on" positions are turned off, and all the "off" positions are turned on

AND	Each position in the array is "ANDed" with the current status of that position on the screen. A position will be turned on only if it is "on" in the array **and** "on" on the screen.
OR	Each position in the array is "ORed" with its corresponding position on the screen. A position is turned on if it is "on" in the array **or** if it is "on" on the screen **or** both.
XOR	Each position in the array is "XORed" with its corresponding position on the screen. A position is turned on only if its status in the array is the **opposite** of its status on the screen.

These options give the PUT@ statement a great deal of flexibility. For example, a figure can appear to blink by PUTting it on the screen with two PUT@ statements that alternate SET and RESET, or by XORing it with the surrounding area. By using OR, you can have two objects intersect and show both objects.

NOTES ON USING GET@ AND PUT@

Always use the same mode when you are GETting and PUTting an array. If you GET@ an array in Character Mode, use a Character Mode PUT@ to put it back on the screen. Otherwise, the PUT@ will produce undefined results.

Except for animation and certain special effects like blinking, it is **usually better to use Character Mode** when GETting and PUTting arrays. This Mode is faster because it moves from character position to character position instead of from graphic block to graphic block. It is also easier to define than Graphics Mode.

A handy procedure for GETting an array, wiping it off the screen, and PUTting it back is as follows:

```
10 GET@ (X1,Y1)-(X2,Y2), A%
20 LINE (X1,Y1)-(X2,Y2), " ",BF
30 PUT@ (X3,Y3)-(X4,Y4),A%
```

Line 20 wipes out the array with blank spaces, " ".

Important. When using GET@ and PUT@, it is important to remember that Character Mode and Graphics Mode simply define two ways to divide up the screen. Any alpha-numeric characters, graphic symbols, lines, or boxes inside the defined space can be saved with a GET@ in either mode.

DIMENSIONING GRAPHIC ARRAYS

The array names in PUT@ and GET@ statements have to be dimensioned with enough "array elements" to hold all the indicated data. Otherwise, an ILLEGAL FUNCTION CALL is returned.

To determine the number of array elements in either Character Mode or Graphic Mode, you **first have to determine the number of storage bytes** required.

Once you have determined the number of bytes required, you can determine the **number of storage elements** by the following formulas:

Type of Array	Number of Elements
Integer Array (%)	bytes/2
Single Precision Array (!)	bytes/4
Double Precision Array (#)	bytes/8

Storage Bytes. In Character Mode, each character uses a single byte for storage. For example, to GET@ one line, 64 bytes of storage are needed. GETting an entire screen requires 1024 bytes, or GETting an 8 by 10 character area requires 80 bytes.

In Graphics Mode, you add the total number of characters, divide by 8, and add 2 to this to determine the number of bytes. The formula is: Bytes = (Characters/8) + 2.

MORE EXAMPLES OF GRAPHICS PROGRAMS

RESHAPING AN ARRAY

The following program is an example of "reshaping an array." We GET@ an array that is in one shape of box and PUT@ it into another shape of box. This technique can be made to work well in Character Mode, but it is very difficult to accomplish in Graphics Mode.

Program	Explanation
10 CLS	clears screen
20 PRINT@ 0, "HANG IN THERE!"	start message at Character Position 0
30 FOR Z = 1 TO 200:NEXT	timing loop
40 DIM A%(7)	dimension array
50 GET@ (0,0)-(15,0), A%	GETting array and calling it A%
60 CLS	clear screen to wipe out message
70 PUT@ (0,0)-(0,15),A%	PUTting array back in different shape
80 GOTO 80	creates loop to prevent READY prompt from wiping out our message (try it without this line)

ROCKETS

Here we have two programs that demonstrate some of the differences between Graphics Mode and Character Mode. In the first program, the rocket moves smoothly up the side of the screen because it is in Graphics Mode. In the second program, we switch to Character Mode and the movements are jerky, but much faster.

Program	Explanation
10 'ROCKET IN GRAPHICS MODE	remark
20 CLS	clear screen
30 LINE (3,1)-(3,2), SET	draws rocket
40 LINE (2,3)-(4,4), SET,BF	
50 LINE (1,4)-(1,5), SET	
60 LINE (5,4)-(5,5), SET	
70 DIM A%(2)	dimension array
80 GET@ (1,1)-(5,6),G,A%	GETting array
90 CLS	clear screen
100 FOR Y = 42 TO 1 STEP – 1	so rocket will start at bottom of screen and move up
110 PUT@ (1,Y)-(5,Y + 5),SET,A%	PUTting array on bottom of screen
120 NEXT Y	creates movement
130 CLS	clears screen
140 GOTO 100	let's do it again, and again, and again...

The difference between Graphic Mode and Character Mode is somewhat like the difference between today's movies and silent pictures. See if you can find where we changed the above program to put it in **Character Mode:**

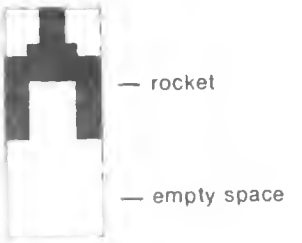
Program

```
10 'ROCKET IN CHARACTER MODE
20 CLS
30 LINE (3,1)-(3,2), SET
40 LINE (2,3)-(4,4), SET, BF
50 LINE (1,4)-(1,5), SET
60 LINE (5,4)-(5,5), SET
70 DIM A%(7)
80 GET@(0,0)-(3,3),A%
90 CLS
100 FOR Y = 12 TO 1 STEP - 1
110 PUT@ (0,Y)-(3,Y + 3),A%
120 NEXT Y
130 CLS
140 GOTO 100
```

To slow this down a bit, try adding a FOR/NEXT timing loop between Lines 110 and 120.

Animation Hint. The program PUTs the graphic image of a rocket ship on the screen at a specific location on the lower-left corner of the screen. Then it creates movement by progressively PUTting it at a higher location. But what makes the image disappear from its old location each time it is PUT at a new location?

The answer to this question is that the program GETs an array that is bigger than the image of the rocket ship. The array that the program GETs has a block of blank spaces beneath the rocket image. It looks something like this:



Array A%

Each time the array is moved, the image at the old location is wiped out by the empty space.

One problem with this, however, is once the rocket reaches the top of the screen, movement stops. The image is frozen on the screen. To take care of this, the above program clears the screen with a CLS.

In Character Mode you could wipe out the rocket image by PUTting a second array on top of the first array. This second array would contain blank spaces (" "). In **Graphic Mode** the solution is similar. First, you GET a second array that is an empty space and then PUT it on top of the first array.

FLASHING SHIP

This program draws the image of a ship on the screen. In **Graphics Mode**, it GETs the image in array A% and clears the screen. Then, it repeatedly PUTs the image back on the screen at a different location **using "XOR" as the action indicator**. This technique causes the image to flash on and off.

Program

```
5 CLS
10 DIM A%(50)
20 LINE (3,1)-(3,2), SET: LINE (2,3)-(4,4), SET, BF: LINE
   (1,4)-(1,5), SET: LINE (5,4)-(5,5), SET
30 GET@ (1,1)-(5,6), G, A%
40 CLS
50 PUT@ (20,20)-(24,25),XOR,A%
60 FOR T = 1 TO 50: NEXT
70 GOTO 50
```

EXAMPLES OF ADVANCED GRAPHICS

These examples should give you an idea of some of the more advanced possibilities of LEVEL III graphics. Try them out to see if you can uncover their techniques.

PROGRAM: FADING BOXES

```
10 DIM A%(600), B%(600)
20 CLS: FOR T = 1 TO 4: LINE (RND(127),RND(47))-(RND(127)
   RND(47)),SET,B:NEXT
30 F$ = " ★ ":X = RND(63):IF X>32 THEN M = 64 – X ELSE M = X
40 FOR I = 1 TO M: GET@(1,0)-(X,15),A%:GET@ (X,0)-(62,15),B%
50 LINE (X,0)-(X + 1,15),F$,B:F$ = " "
60 PUT@(0,0)-(X – 1,15),A%:PUT@(X + 1,0)-(63,15),B%:NEXT:
   GOTO 20
```

PROGRAM: SAILING SHIPS

```
10 CLS
20 LINE (3,1)-(3,2), SET:LINE (2,3)-(4,4), SET, BF:
   LINE (1,4)-(1,5), SET:LINE (5,4)-(5,5), SET
50 DIM A%(2):GET@(1,1)-(5,5),G,A%
60 CLS:DIMB%(2)
70 LINE (0,2)-(1,2), SET:LINE (2,1)-(4,3), SET, BF:
   LINE (4,0)-(5,0), SET:LINE (4,4)-(5,4), SET
110 GET@ (0,0)-(5,4),G,B%
120 CLS:X = 120:Y = 41'
130 IF Y = 0 OR X = 0 THEN 120
140 D = RND(2):IF D = 1 THEN 190
150 S = RND(15):IF X – S<0 THEN S = X
160 FOR X = X TO X – S STEP – 1
170 PUT@(X,Y)-(X + 5, Y + 4),SET,B%
180 NEXT X:X = X + 1:GOTO 130
190 S = RND(5):IF Y – S<0 THEN S = Y
200 FOR Y = Y TO Y – S STEP – 1
210 PUT@(X,Y)-(X + 4,Y + 4),SET,A%
220 NEXT Y:Y = Y + 1:GOTO 130
```

PROGRAM: LEVEL III BRAGS ABOUT ITSELF

```
5 CLEAR 300
10 READ A$
20 IF A$ = "END" THEN RESTORE:GOTO 10
30 FOR W = 1 TO 5:CLS
40 PRINT@470,"LEVEL 3 DOES IT!":
   PRINT@470 + 64,STRING$(16,95);
50 FOR X = 1 TO LEN(A$)
60 LINE (X – 1,X – 1)-(64 – X,16 – X),MID$(A$,X,1),B
70 NEXT
80 FOR T = 1 TO 90:NEXT
90 NEXT:GOTO 10
100 DATA HEY THERE, LISTEN, EVER SEEN, A BETTER,
   BASIC, NO WAY  ONLY
110 DATA LEVEL 3, DOES IT, AND MORE, !!!!!!, END
```

PROGRAM: RADAR SCREEN

```
10 FOR R = 15 TO 1 STEP  – 1
20 CLS
30 FOR Y = 0 TO 47 STEP 47
40 FOR X = 0 TO 127 STEP R
50 LINE (64,24)-(X,Y),SET
60 NEXT X,Y
70 FOR X = 0 TO 127 STEP 127
80 FOR Y = 0 TO 47 STEP R
90 LINE (64,24)-(X,Y),SET
100 NEXT Y,X
110 NEXT R
```

USING LEVEL III BASIC

SECTION THREE: STRINGS, USER DEFINED FUNCTIONS, AND MACHINE LANGUAGE SUBROUTINES

INCLUDING:

- NEW MID$ CAPABILITY
- SEARCHING STRINGS WITH INSTR
- DEFINE YOUR OWN FUNCTIONS
- MACHINE LANGUAGE USER ROUTINES

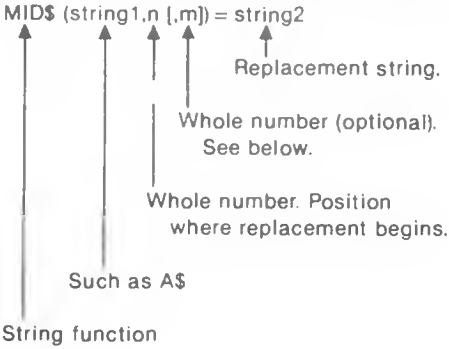
This section contains a number of "miscellaneous" features of LEVEL III, some of which you are sure to appreciate. If you've had any experience using string functions you'll find LEVEL III's MID$ and INSTR very useful. Defining your own functions is, well, shouldn't everybody?

NEW MID$ CAPABILITY

In Level II Basic, the MID$ function is used to return a **substring** of a given string. The example program in your Level II Manual (see page 5/6) returns the "exchange" or first three numbers of a phone number.

In LEVEL III BASIC, MID$ can be used on the **left side** of an assignment statement as well as the right side. This allows you to use MID$ to **replace a portion of one string with another string**.

The format of MID$ on the left side of an equation is:



MID$

The characters in string1, beginning at position n, are replaced by the characters in string2. The m is optional; it refers to the number of characters from string2 that will be used in the replacement. If m is omitted, all of string2 will be used

For example, the following program replaces the MO in "KANSAS CITY, MO" with a KS:

```
10 A$ = "KANSAS CITY, MO"
20 MID$(A$, 14) = "KS"
30 PRINT A$
RUN
KANSAS CITY, KS
```

When using this function, you should be aware that MID$ literally replaces **characters in string1 with characters in string2**. Therefore, the replacement of characters will never go beyond the original length of string1. You cannot replace the MO in A$ above with KANSAS. The result would be KANSAS CITY, KA!

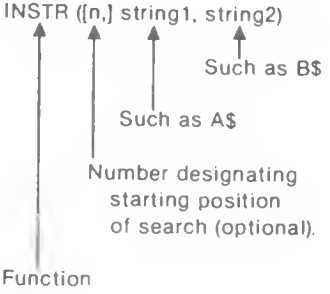
For the sake of further illustration, suppose that you wanted to replace the spelling J-O-H-N with J-O-N. The following program might seem to be appropriate:

```
10 A$ = "JOHN"
20 MID$(A$,3) = "N"
30 PRINT A$
```

but the result is JONN. This can be corrected simply by adding a space in string2 of line 20: 20 MID$(A$,3) = "N ". Now JOHN will become JON.

THE INSTR FUNCTION

This is a new string function not found in Level II Basic. It allows you to **search a string for a specified substring**. The format of INSTR is:



INSTR

INSTR searches string1 for a substring that matches string2. When a match is found, INSTR returns the starting position of the match. The n is optional. It designates the starting position of the search. If n is omitted, the search starts with the first character in string1.

If no match is found, **or** if n is greater than the length of string1, **or** if string1 is null, INSTR returns a zero (0).

Example. The following example shows INSTR being used with and without the optional n. It searches the string "ABCDEABCDE" for the substring "BC":

Program	**Explanation**
10 A$ = "ABCDEABCDE"	
20 B$ = "BC"	
30 PRINT INSTR (A$,B$)	Without n
40 PRINT INSTR (3,A$,B$)	With n
RUN	
2	Result of line 30
7	Result of line 40

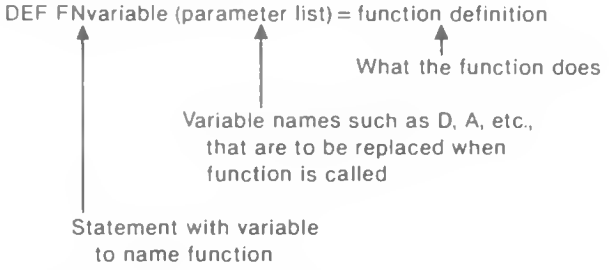
Note that once a substring is found the search is discontinued. INSTR does not continue searching for additional substrings.

Important: The INSTR function eliminates the need for using "instring subroutines" as described on page 5/9 of your Radio Shack **Level II BASIC Reference Manual**. INSTR provides the same capability, it is much easier and faster to use.

DEFINING YOUR OWN FUNCTIONS

Often a program will contain a particular operation or function that repeated several times. By using the DEF FN statement you can defir your own functions and save both time and memory. Instead of writir out the entire formula each time it is used, you only need to do a "fun tion call."

DEF FN. This statement is used to name and define user functions. Tl format of the statement is:



The variable name, which is any legal variable name, is tacked on to DE FN to become the name of the function. An example would be: DEF FN

The parameter list is one or more variables or string variables separat by commas and enclosed in parentheses. These variables are **variabl you want to replace** when the function is called. Seems a bit co plicated, but the examples that follow should help clarify it for you.

The function definition is an expression that performs the necessa operation. It could be something like: A/4 + E * 25. Variable names th appear in the function definition do not affect other program variabl that happen to have the same variable names.

Example. In the following example, a function is defined that adds the second power of one number to the third power of another.

```
10 DEF FNA (D,E) = D↑2 + E↑3
20 A = 1,B = 2
30 PRINT FNA (A,B),FNA(2,3),FNA(4,3),FNA(5,2)
RUN
9          31          43          33
```

We defined a function called DEF FNA that adds the second power of D to the third power of E. Since we want to be able to change the values of D and E, they are included in the parameter list.

The DEF FN statement can also **define a string function**, as in this example:

Program

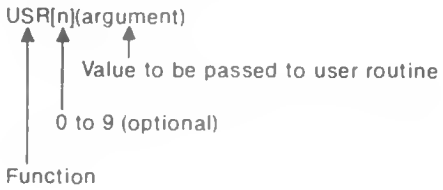
```
5 CLEAR 500                    just so we'll have ample string space
10 DEF FNST$(A$,B$) = A$ + ", " + B$
20 INPUT "STATE";ST$
30 INPUT "CITY";CT$
40 F$ = FNST$(CT$,ST$)
50 PRINT F$
RUN
STATE? NEBRASKA
CITY? ALLIANCE
ALLIANCE, NEBRASKA
```

NOTE: The variable name in the second example ends with a dollar sign($). Function name variable, like other kinds of variables, must indicate the type of value that is to be returned. Thus a function name variable may end with a $ (string), # (double precision), %(integer), or ! (single precision). The default is single precision (!).

MACHINE LANGUAGE USER ROUTINES

In LEVEL III BASIC, the USR function has been expanded so that 10 different machine language user routines can exist in memory at the same time. As with Level II Basic, the routines may be assembled with the TRS-80 Editor/Assembler and loaded with the SYSTEM command. Or, they can be POKEd into memory. It is **no longer necessary** to POKE the starting address of a user routine into memory. The DEFUSR statement is provided for this purpose.

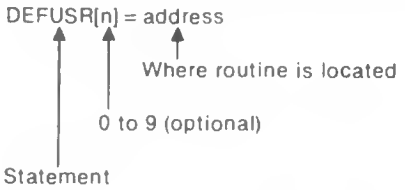
USR Function. The new format for this function is:



USR

The optional [n] is a number from 0 to 9. It is used to label the different machine language user routines. If it is not included, the "0" is assumed making the function the same as USR0.

DEFUSR Statement. This statement tells BASIC what the starting address of a routine is. The format of DEFUSR is:



DEFUSR

The n is a number from 0 to 9 that should correspond with the number of the user routine. If it is omitted, it is assumed to be a 0.

Example. A user routine called USR3 has been POKEd into memory beginning at address 28000. Before calling the user routine, the program must execute the statement:

```
DEFUSR3 = 28000
```

A calling statement for this routine might look like this:

```
A = USR3(B)
```

If a user routine is called before the corresponding DEFUSR statement has been executed, an ILLEGAL FUNCTION CALL error results.

It is still possible to POKE the starting address of USR0 into memory, as described in Chapter 8 of your Radio Shack **LEVEL II BASIC Reference Manual**.

- **THE LINE INPUT STATEMENT**
- **INPUT#LEN AND LINE INPUT#LEN**
- **NEW LOAD AND SAVE COMMANDS**
- **TURNING OFF THE SYSTEM CLOCK***
- **OUTPUT TO RS-232 PORT****

The I/O features described in this section include two powerful additions to Level II Basic's INPUT statement, and a much improved way to load and save cassette tapes.

The INPUT features allow you to enter INPUT with commas or other punctuation without enclosing the whole input in quotation marks. You can also put a time limit on inputs, which adds a whole new dimension to game programs and educational programs.

By using the new LOAD and SAVE commands to replace Level II's CLOAD and CSAVE commands, you eliminate much of the agony of loading and saving programs on tape.

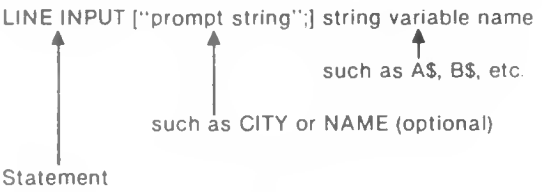
"Output to RS-232 Port" and "Turning Off the System Clock" are two things you don't have to even think about unless you have an Expansion Box as part of your TRS-80 System. They do, however, demonstrate some of the foresight of LEVEL III BASIC.

*Requires TRS-80 Expansion Box

**Requires RS-232 Port

THE LINE INPUT STATEMENT

The LINE INPUT statement functions much like INPUT except that it has some added capability. The format of this statement is:



LINE INPUT

As you can see from the format, LINE INPUT is frequently used for input ting string literals. Unlike INPUT, this statement will assign a strin variable name to the entire line of input. Every character typed up t [ENTER] will be part of the string. This includes commas, colons quotes, or leading spaces.

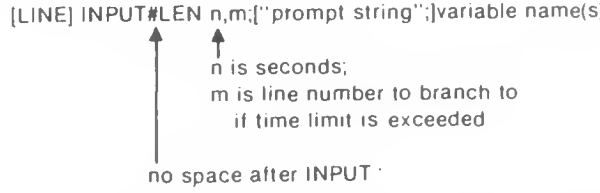
Another aspect of LINE INPUT is that it doesn't automatically print a question mark (?) when it is executed, as do INPUT statements. If yo want a question mark, you simply make it a part of the prompt strin such as "CITY?".

Example. When the following LINE INPUT statement is executed, i displays CITY, STATE: as the prompt string. You can then enter a response that includes a comma such as ALLIANCE, NEBRASKA. You response will be assigned to the string literal, CS$.

```
10 LINE INPUT "CITY,STATE: ";CS$
20 PRINT CS$
RUN
CITY, STATE: ALLIANCE, NEBRASKA
ALLIANCE, NEBRASKA
```

INPUT#LEN AND LINE INPUT# LEN

By adding #LEN to an INPUT or LINE INPUT statement, you can impose a limit on the length of time allowed for a response to the INPUT prompt. And you don't need an Expansion Box because the timing is done with software. The format of this is:



The "n" is the time limit in seconds. You can use up to 8000 seconds, which is 2 hours, 13 minutes and 20 seconds! The "m" is the line number you want the program to go to if the time limit is exceeded. It could branch to a PRINT statement that says something like, "SORRY, DUMMY, YOU BLEW IT!!!"

Other than tacking #LEN on INPUT and adding n and m, this statement works the same as any other INPUT or LINE INPUT statements.

Example. This program selects random addition problems and gives you just four seconds to answer.

Program

```
5 S = 4
10 X = RND(100)
20 Y = RND(100)
30 PRINT X;" + ";Y;" = ";
40 INPUT#LEN S,100;Q
50 IF Q = X + Y THEN PRINT "SMART" ELSE PRINT "DUMB"
60 GOTO 10
100 PRINT "SLOW!"
110 GOTO 10
```

NEW LOAD AND SAVE COMMANDS

In Level II Basic, the CLOAD and CSAVE commands are not always reliable because they depend on an exact volume setting on the tape recorder. LEVEL III BASIC replaces these commands with new commands to eliminate the problem. Use LOAD and SAVE exactly as you used CLOAD and CSAVE.

LOAD may be used to load a tape that was previously saved with a Level II Basic CSAVE command. Likewise, a tape that is SAVEd in LEVEL III BASIC may be subsequently CLOADed in Level II Basic, but the volume setting will again become critical.

LOAD and SAVE commands can be aborted with the [BREAK] key. Just remember that when [BREAK] is used during a SAVE the partial tape will be of no use. When [BREAK] is used during a LOAD you will have LOADed part of the program in your Ram memory. Type NEW before you try entering anything else.

LOAD? can be used to compare a program stored on cassette with one presently in the computer's memory. This is the same as the Level II Basic CLOAD? command.

TURNING OFF THE SYSTEM CLOCK*

If your system includes an Expansion Box, the system clock is "on" while LEVEL III BASIC is running. Tape operations are vulnerable to interruptions from the system clock. Therefore, the clock should be turned off before a SYSTEM, INPUT# – 1, or PRINT# – 1 command and turned back on afterward. (Don't worry about LOAD and SAVE—they turn the clock off and on automatically.)

The command that turns the clock off is CMD"T" and the command that turns it back on is CMD"R." These commands may also be used as program statements.

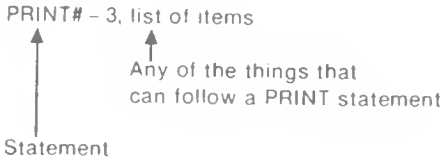
Example. Here we turn the clock off, read values from tape, and turn the clock back on.

```
100 CMD"T"
120 INPUT# – 1,X,Y,Z
130 CMD"R"
```

*For use with TRS-80 System that includes an Expansion Box

OUTPUT TO RS-232 PORT* *

To output to an RS-232 port, LEVEL III provides a PRINT# – 3 statement, which is used just like the PRINT statement. The format of this statement is:



PRINT#-3

This makes it easy to output to a line-printer or any other device you hook up to the RS-232 port. Input from the RS-232 port still requires the use of machine language routines.

The first character sent by BASIC to the RS-232 port causes the RS-232's UART to initialize using the switches set on the RS-232. In order to override this default initialization, send a dummy character to the RS-232 port and then re-initialize using a machine language subroutine.

* *Requires RS-232 Unit

INDEX TO STATEMENTS, COMMANDS, AND FUNCTIONS

INDEX